

## Содержание (сводка)

	Введение	23
1	С чего начинаются хорошие программы. <i>О пользе качественного проектирования</i>	35
2	Дайте им то, что они хотят. <i>Сбор требований</i>	87
3	Я тебя люблю, ты мой идеал... Теперь изменись. <i>Изменение требований</i>	141
4	Программы для реального мира. <i>Анализ</i>	173
5	Часть 1. Все течет, все меняется. <i>Качественное проектирование</i>	225
	Вставка: ОО-КАТАСТРОФА	249
	Часть 2. Зарядка для программ. <i>Гибкость программы</i>	261
6	«Меня зовут Арт... И я архитектор». <i>Решение больших задач</i>	305
7	Навести порядок в хаосе. <i>Архитектура</i>	347
8	Не стремитесь к оригинальности. <i>Принципы проектирования</i>	397
9	Программы пишутся для заказчика. <i>Итерации и тестирование</i>	443
10	Все вместе. <i>Жизненный цикл ООАП</i>	501
	Приложение I. <i>Остатки</i>	571
	Приложение II. <i>Добро пожаловать в Объективль</i>	589

## Содержание (настройка)

**Введение**

**Ваш мозг думает об ООАП.** Вы сидите за книгой и пытаетесь что-нибудь выучить, но ваш мозг считает, что вся эта писанина не нужна. Ваш мозг говорит: «Выгляни в окно! На свете есть более важные вещи, например сноуборд». Как заставить мозг думать, что ваша жизнь действительно зависит от объектно-ориентированного анализа и проектирования?

Для кого написана эта книга?	24
Мы знаем, о чем вы думаете	25
Метапознание: наука о мышлении	27
Вот что сделали МЫ	28
Что можете сделать ВЫ	29
Примите к сведению	30
Благодарности	33

О пользе качественного проектирования

1

**С чего начинаются хорошие программы**

**Так как же на самом деле пишутся хорошие программы?** Всегда сложно понять, с чего следует начинать. Делает ли приложение то, что ему положено делать? И как насчет таких тонкостей, как дублирование кода, — ведь это всегда плохо, верно? Да, определить, **над чем следует работать в первую очередь**, бывает нелегко, к тому же еще нужно не запортить все остальное по ходу дела. Но пусть вас это не тревожит! К тому моменту, когда вы дочитаете эту главу, вы **будете знать, как пишутся хорошие программы**, а ваш подход к разработке приложений навсегда изменится. И наконец-то вы поймете, почему ООАП — это такое слово, которое не стыдно произносить в приличном обществе.

Откуда я знаю, с чего начинать? В каждом новом проекте, над которым я работаю, у всех разные мнения по поводу того, что делать в первую очередь. Иногда я угадываю верно, иногда приходится переделывать все приложение из-за того, что я начал не с того. А я просто хочу писать хорошие программы! Так с чего нужно начать в приложении Рика?

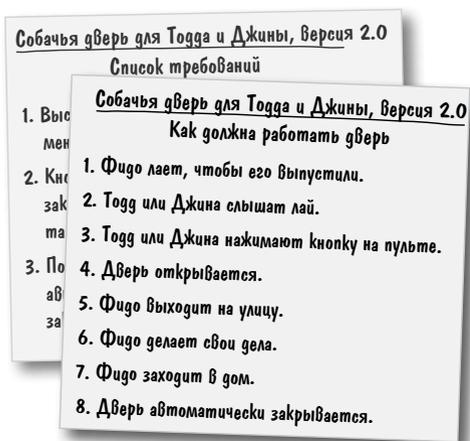


Рок-н-ролл навсегда!	36
Новенькое приложение Рика...	37
Что бы вы изменили В ПЕРВУЮ ОЧЕРЕДЬ?	42
Хорошая программа... сочетание нескольких факторов	46
Хорошая программа за 3 простых шага	47
Пробный запуск	57
Ищем проблемы	59
Анализируем метод search()	60
Теперь обновите свой код	64
Обновление класса Inventory	66
Готовимся к следующему пробному запуску	67
Возвращаемся к приложению Рика...	69
Продолжаем совершенствовать структуру кода	70
Убедимся в том, что класс Inventory хорошо спроектирован	71
Последнее тестирование (проверка готовности кода к повторному использованию)	80
Цель ООАП — написание хороших программ, а не формализация рабочего процесса!	83
Ключевые моменты	64

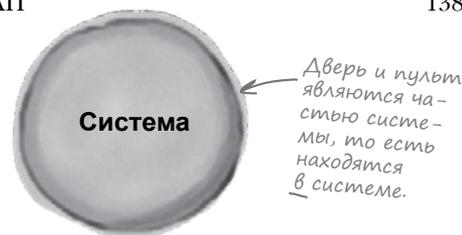
# 2 Сбор требований

## Дайте им то, что они хотят

**Хорошо, когда заказчик доволен.** Вы ведь уже знаете, что написание хороших программ начинается с того, что вы проверяете, работает ли программа так, как хочет заказчик. Но как узнать, чего на самом деле хочет заказчик? И как убедиться в том, что заказчик вообще знает, чего он хочет? На помощь приходят хорошо составленные требования. В этой главе вы узнаете, как выполнить требования заказчика и предоставить ему именно то, чего он хотел. После этого на всех ваших проектах можно будет ставить «знак качества», а вы сделаете еще один большой шаг к заветной цели — написанию хороших программ... раз за разом.



Вам поручена новая задача	88
Пробный запуск	91
Создание списка требований	96
Планирование возможных неудач	100
Альтернативные пути для решения проблем	102
(Снова) о вариантах использования	104
Один вариант, три части	106
Проверка требований по варианту использования	110
Все ли на месте?	110
Ну теперь-то можно заняться написанием кода?	113
Автоматическое закрывание двери	114
Нужен новый тест!	115
Пробный запуск, версия 2.0	116
Анализ альтернативного пути	118
Пробный запуск, версия 2.1	121
Приложение работает, заказчики довольны	123
Инструментарий ООАП	138



# 3 Изменение требований

## Я тебя люблю, ты мой идеал... Теперь изменись

**Думаете, вы сделали то, что нужно заказчику? Как бы не так...**

Вы поговорили с заказчиком, собрали требования, записали варианты использования и выдали убойное приложение. Можно расслабиться, верно? Верно... Пока заказчик не решит, что ему нужно **что-то отличное от того, о чем он говорил вам**. То, что вы сделали, ему понравилось, честно, но **теперь это не совсем то, что нужно**. В реальном мире **требования всегда изменяются**; вы должны адаптироваться к этим изменениям и сделать так, чтобы заказчик был доволен.

Вы — герой!	142
А потом был телефонный звонок...	142
Снова беремся за карандаш	144
Единственный постоянный фактор в области анализа и проектирования программного обеспечения	145
Варианты использования должны быть понятными для вас	152
Готовимся к написанию кода...	159
Доработка списка требований	160
А теперь можно снова заняться программированием	161
Кто сказал «гав»?	162
Проверка новой двери	164
Обновление класса двери	169
Упрощение класса пульта	169
Последний пробный запуск	170
Новые инструменты ООАП	172

```

public void pressButton() {
    System.out.println("Pressing the remote control button...");
    if (door.isOpen()) {
        door.close();
    } else {
        door.open();
    }

    final Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        public void run() {
        door.close();
        timer.cancel();
        }}
    , 5000);
}
}

```



Remote.java

# 4 Анализ

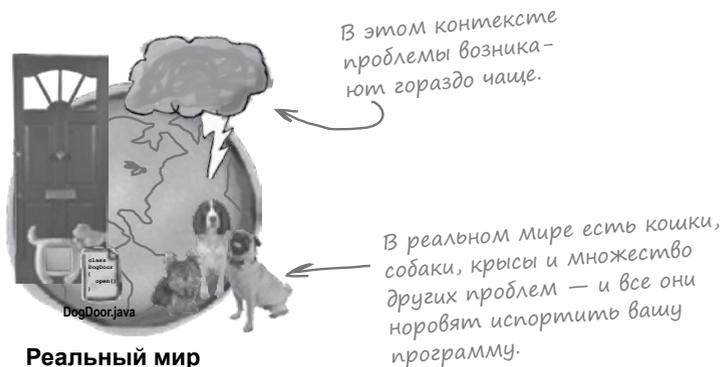
## Программы для реального мира

**Пора переходить к реальным приложениям.** Даже если ваше приложение идеально работает на машине разработки, этого недостаточно: приложения должны работать тогда, когда их используют реальные люди. В этой главе говорится о том, как заставить вашу программу работать в реальном контексте. Вы узнаете, как текстологический анализ преобразовывает вариант использования, над которым вы работали, в классы и методы, которые делают то, что нужно вашим заказчикам. А когда работа будет завершена, вы сможете сказать: «Да, у меня получилось! Моя программа готова к жизни в реальном мире!»

Потом, когда я поделилась с классами и операциями, я обновила свою диаграмму классов.



Одна собака, две собаки, три...	174
Программа работает в определенном контексте	175
Выявление проблемы	176
Планирование решения	177
История двух программистов	184
Делегирование в версии Сэма	188
Сила приложений с низкой связностью	190
Обращайте внимание на существительные в варианте использования	195
От качественного анализа к хорошим классам...	208
Подробнее о диаграммах классов	210
Диаграммы классов — это еще не все	215
И как теперь работает метод recognize()?	217



Реальный мир

Качественное проектирование = Гибкость программного продукта

# 5 (часть I)

## Все течет, все меняется

**Изменения неизбежны.** Какой бы замечательной ни была ваша программа сегодня, завтра ее, скорее всего, придется изменять. И чем труднее вносить поправки в программу, тем сложнее реагировать на изменяющиеся потребности заказчика. В этой главе мы вернемся к старому знакомому, попробуем улучшить существующую программу и увидим, как маленькие изменения создают большие трудности. А при этом мы обнаружим проблему настолько значительную, что для ее решения потребуется глава ИЗ ДВУХ ЧАСТЕЙ!

Струнные инструменты	226
Давайте проверим структуру кода на прочность	226
Вы обратили внимание на абстрактный базовый класс?	229
Как устроены диаграммы классов (снова)	234
Напишем код новой поисковой программы Рика	236
Создание абстрактного класса для спецификаций инструментов	237
Завершение работы над поисковой программой	240

# 5 (вставка)

## 00-КАТАСТРОФА!

Любимая телевикторина Объеквила

Исключение риска	Знаменитые проектировщики	Программные конструкции	Сопровождение и повторное использование	Программные невроты
\$100	\$100	\$100	\$100	\$100
\$200	\$200	\$200	\$200	\$200
\$300	\$300	\$300	\$300	\$300
\$400	\$400	\$400	\$400	\$400

# 5

(часть 2)

Хорошая структура = Гибкость программы

## Зарядка для программ

**А вам когда-нибудь хотелось обрести большую гибкость?**

Если внесение изменений в приложение создает проблемы, скорее всего, это означает, что вашей программе не хватает гибкости и пластичности. Чтобы помочь приложению прийти в хорошую форму, необходимо провести анализ, основательно проработать структуру кода приложения и узнать, как ОО-принципы способствуют ослаблению связности. А в завершение вы увидите, как высокое сцепление способствует ослаблению связности. Звучит заманчиво? Переверните страницу, и мы продолжим наводить порядок в нашем негибком приложении.

Возвращаемся к приложению Рика	262
Присмотримся повнимательнее к методу search()	265
Результаты анализа	266
Проблемы с классами инструментов	269
Но ведь классы создаются ради поведения!	269
Смерть проектировочных решений	274
Субклассы конкретных инструментов	274
Преобразуем плохие структурные решения в хорошие	275
«Двойная инкапсуляция» в программе Рика	277
Динамические свойства инструментов	278
Смотрите: гибкое приложение Рика!	286
А приложение действительно работает?	287
Пробный запуск улучшенной программы Рика	289
Высокое сцепление и одна причина для изменения	298
Когда следует сказать: «Достаточно!»	303
Инструментарий ООАП	304

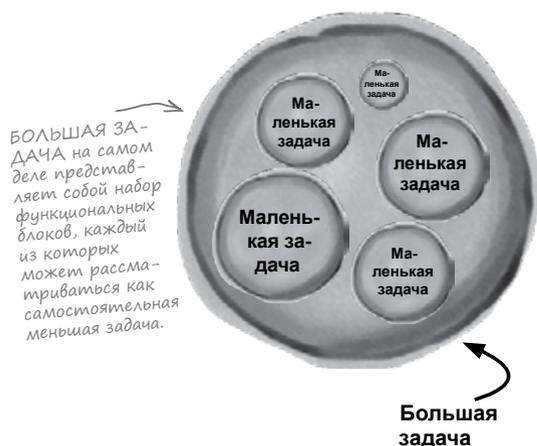
# 6 Решение больших задач

## «Меня зовут Арт... И я архитектор»

### Пора построить нечто **ДЕЙСТВИТЕЛЬНО БОЛЬШОЕ**. Готовы?

В вашей инструментарии ООАП скопилось множество инструментов, но как использовать эти инструменты при построении большого проекта? Возможно, вы этого и не осознали, но у вас имеется все необходимое для решения больших задач. В этой главе мы рассмотрим новые инструменты — анализ предметной области и диаграммы вариантов использования. Но даже эти инструменты основаны на том, что вам уже известно, — на необходимости прислушиваться к мнению заказчика и необходимости понимания того, что вы строите, до начала работы над кодом. Приготовьтесь... Пора поиграть в архитектора.

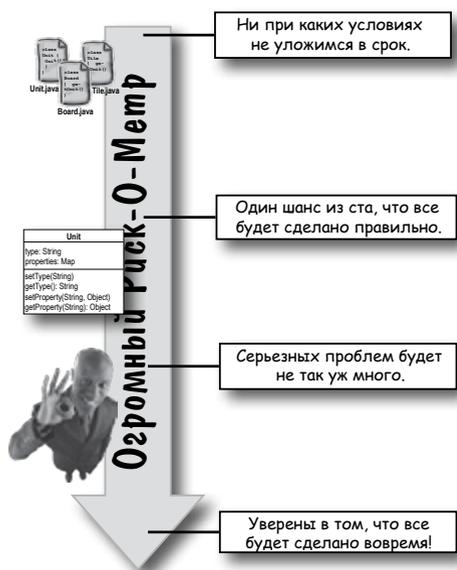
Все дело в подходе к решению большой задачи	307
Так давайте решим БОЛЬШУЮ задачу!	309
Необходимо больше информации	313
Определение функциональных возможностей	316
Используем диаграммы вариантов	322
Маленький субъект	327
Субъекты тоже люди (но не всегда)	328
Займемся анализом предметной области!	333
Разделяй и властвуй	335
Не забывайте, кто ваш заказчик	339
Что такое «паттерны проектирования»? И как их использовать?	341
Сила ООАП (и немного здравого смысла)	344
Инструментарий ООАП	346



## Архитектура

### Навести порядок в хаосе

Любая работа с чего-то начинается, но вам стоит выбрать **правильное что-то!** Вы уже знаете, как разбить приложение на множество мелких задач, но это означает, что мелких задач будет действительно МНОГО. В этой главе мы поможем вам разобраться, с чего начать и как избежать потерь времени на движение в ошибочном направлении. Пришло время взять все эти мелкие детали, разложенные на вашем рабочем месте, и понять, как собрать из них четко структурированное, хорошо спроектированное приложение. Попутно вы узнаете о трех главных вопросах архитектуры и поймете, что «Риск» — это не только интересная настольная игра из 1980-х годов.



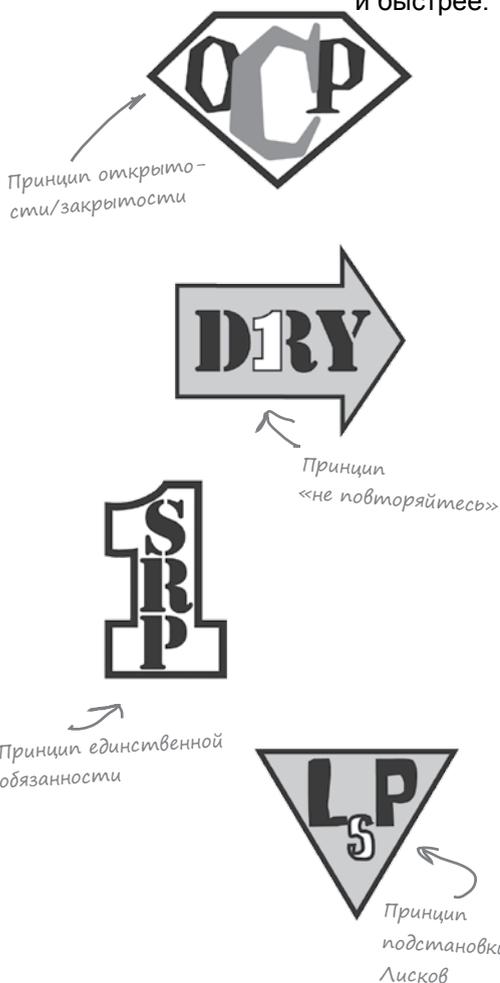
Слегка растерялись?	348
Нам нужна архитектура	350
Начнем с функциональности	353
Но что из этого считать самым важным?	353
Три основных вопроса архитектуры	356
Сценарии способствуют сокращению рисков	365
Классы Tile и Unit	372
Больше порядка, меньше хаоса	374
Чем займемся теперь?	375
Юниты для конкретной игры... что это означает?	376
Возвращаемся к общности	379
Анализ общности: путь к гибкости кода	385
Сокращение рисков помогает писать хорошие программы	395
Ключевые вопросы	396

## Принципы проектирования



### Не стремитесь к оригинальности

**Имитация — самая искренняя форма проявления ума.** Ничто не доставляет столько удовлетворения, как создание совершенно нового, оригинального решения задачи, которая не давала вам покоя несколько дней. Пока не выяснится, что кто-то уже решил ту же задачу задолго до вас, да еще и сделал это лучше вас! В этой главе рассматриваются принципы проектирования, которые были созданы за прошедшие годы, и то, как они помогают вам стать более квалифицированным программистом. Откажитесь от мысли «сделать по-своему», в этой главе вы узнаете, как сделать умнее и быстрее.



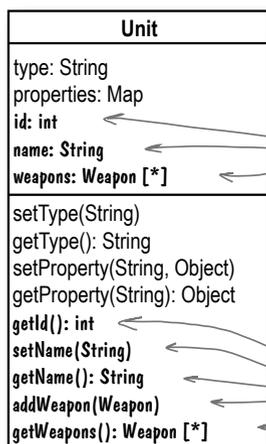
Кратко о принципах проектирования	398
Принцип № 1: принцип открытости/закрытости	399
ОСР шаг за шагом	401
Принцип № 2: не повторяйтесь	404
DRY: ОДНО требование в ОДНОМ месте	406
Принцип № 3: принцип единственной обязанности	412
Выявление множественных обязанностей	414
От множественных обязанностей к единственной	417
Принцип № 4: принцип подстановки Лисков	422
Пример неправильного использования наследования	423
Принцип LSP выявляет скрытые проблемы в структуре наследования structure	424
«Замена базового типа субтипом не должна отразиться на работе программы»	425
Нарушение LSP усложняет код	426
Делегирование функциональности другому классу	428
Использование композиции для объединения поведения других классов	430
Агрегирование: композиция с продолжением существования	434
Агрегирование и композиция	435
Наследование — всего лишь одна из возможностей	436
Ключевые моменты	439
Инструментарий ООАП	440

# 9 Итерации и тестирование

## Программы пишутся для заказчика

Пришло время показать заказчику, что вы действительно беспокоитесь о его интересах. Назойливое начальство? Обеспокоенные заказчики? Люди, которые постоянно спрашивают: «Ну что, успеете к сроку?» Никакой хорошо структурированный код не порадует вашего заказчика; вы должны показать ему что-то работающее. А теперь, когда вы овладели солидным инструментарием ОО-программирования, пора узнать, как доказать заказчику, что его программа действительно работает. В этой главе мы рассмотрим два способа углубленного анализа функциональности вашего продукта, после которого заказчик с чувством скажет: «Да, вы определенно идеально подходите для этой работы!»

Инструментарий постепенно заполняется	444
Но программу-то вы пишете для ЗАКАЗЧИКА!	445
Углубление итераций: два основных варианта	447
Функционально-ориентированная разработка	448
Сценарно-ориентированная разработка	449
Два метода разработки	450
Использование функционально-ориентированной разработки	453
Анализ функциональной возможности	454
Написание тестовых сценариев	457
Воспользуемся решением, ориентированным на общность	470
Сопоставьте тесты с разработанной структурой	472
Изучаем тестовые примеры...	474
Представьте результат заказчику	480
До настоящего момента мы занимались контрактным программированием	482
Контрактное программирование основано на доверии	483
Перемещение юнитов	492
Ключевые моменты	495
Инструментарий ООАП	498



Все свойства, общие для всех юнитов, представляются специальными переменными, не входящими в контейнер Map.

Сэм решил, что идентификатор будет задаваться в конструкторе Unit, поэтому метод setId() не нужен.

Для каждого из новых свойств создается набор методов.

# 10

## Жизненный цикл ООАП

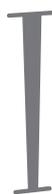
### Все вместе

**Далеко ли до цели?** Мы изучили множество способов совершенствования программных продуктов, но пришло время собрать все воедино. Да, это именно тот момент, которого вы так долго ждали: мы возьмем все, о чем вы узнали из книги, и покажем, что в действительности это были лишь части общего процесса, который можно использовать вновь и вновь для написания хороших программ.

Разработка в стиле ООАП	502
Жизненный цикл проекта в стиле ООАП	503
Задача	506
Начало итераций	521
Подробнее о представлении линий метро	523
Использовать класс Line или не использовать класс Line...	532
Достопримечательности (класса) метро Объектвиля	538
Защита ваших классов (и классов клиентов)	541
Возвращаемся к фазе требований...	551
Все внимание коду, потом все внимание заказчикам	553
Итеративный метод упрощает решение	557
Как выглядит маршрут?	562
Увидеть Объектвиль собственными глазами	566
Нужна ли итерация №3?	569
Путешествие продолжается...	570



## Приложение I. Остатки



### Десять главных тем (не рассмотренных в книге)

**Хотите верьте, хотите нет, но это еще не все.** Да, позади осталось более 550 страниц, и кое-что в них не уместилось. И хотя каждая из последних десяти тем заслуживают разве что краткого упоминания, мы решили выдать вам на дорогу чуть больше информации по каждой из них. Теперь вам будет о чем поговорить во время рекламных пауз... да и кто не любит время от времени поговорить о ООАП?

А когда закончите читать это приложение, останется еще одно... И немного рекламы... А потом — действительно всё. Честное слово!

1. Отношения «является» и «содержит»	572
2. Форматы вариантов использования	574
3. Антипаттерны	577
4. Карты CRC	578
5. Метрики	580
6. Диаграммы последовательности	581
7. Диаграммы состояния	582
8. Модульное тестирование	584
9. Стандарты программирования и удобочитаемость кода	586
10. Рефакторинг	588



**Антипаттерны**  
 Антипаттерны являются противоположностью паттернов проектирования: это распространенные ПЛОХИЕ решения. Старайтесь своевременно распознавать ловушки и избегать их.

**Класс: DogDoor**

*Описание: представляет физическую собачью дверь. Предоставляет интерфейс к оборудованию, которое управляет дверью.*

Обязанности:

Название	Сотрудник
Открыть дверь	
Закрыть дверь	

*У этих операций классов-сотрудников нет.*

*Здесь записываются как операции, выполняемые классом самостоятельно, так и операции, в выполнении которых участвуют другие классы.*

# II

## Приложение II. Добро пожаловать в Объектвиль

### Говорим на языке ООП

**Приготовьтесь к путешествию в другую страну.** Пришло время навестить Объектвиль — место, где объекты делают то, что им положено, все приложения хорошо инкапсулированы (вскоре вы узнаете, что это означает), а программные структуры обеспечивают простоту расширения и повторного использования. Но прежде чем браться за дело, необходимо кое-что знать заранее — в частности, овладеть кое-какими языковыми навыками. Не беспокойтесь, это не займет много времени. Вы и не заметите, как начнете говорить на языке ООП так, словно вы уже давно живете в элитном районе Объектвиля.

Добро пожаловать в Объектвиль	590
UML и диаграммы классов	591
Наследование	593
И еще полиморфизм...	595
И наконец, инкапсуляция	596
Теперь кто угодно может задать значение speed напрямую	596
И из-за чего столько шума?	597

